Claims

1.   A method of real-time shadow generation in computer graphical representation of a scene, the method comprising

- defining an eye's frustum based on a desired view of the scene;

- defining a location of a light source illuminating at least a portion of the scene;

- generating a trapezoid to approximate an area, E, within the eye's frustum in the post-perspective space of the light, L;

- applying a trapezoidal transformation to objects within the trapezoid into a trapezoidal space for computing a shadow map; and

- determining whether an object or part thereof is in shadow in the desired view of the scene utilising the computed shadow map.

2.   The method as claimed in claim 1, wherein generating the top and base lines l_t and l_b respectively, of the trapezoid to approximate E in L, comprises

- computing a centre line l, which passes through centres of the near and far planes of E;

- calculating the 2D convex hull of E;

- calculating l_t that is orthogonal to l and touches the boundary of the convex hull of E;

- calculating l_b which is parallel to l_t and touches the boundary of the convex hull of E.

3.   The method as claimed in claim 1, wherein, in the case that the centres of the far and near planes of E are substantially coincident, a smallest box bounding the far plane is defined as the trapezoid.

4.   The method as claimed in claims 1 or 2, wherein generating the side lines of the trapezoid to approximate E in L comprises

- assigning a distance d from the near plane of the eye's frustum to define a focus region in the desired view of the scene;

- determining a point p_L in L that lies on l at the distance d from the near plane of the eye's frustum;

- computing the position of a point q on l, wherein q is the centre of a projection to map the base line and the top line of the trapezoid to $y = -1$ and $y = +1$ respectively, and to map p_L to a point on $y = \xi$, with ξ between -1 and +1; and

-constructing two side lines of the trapezoid each passing through q, wherein each sideline touches the 2D convex hull of E on respective sides of l.

5.   The method as claimed in claim 4, wherein $\xi = -0.6$.

6.   The method as claimed in claim 4, wherein the desired point ξ is determined based on an iterative process that minimizes wastage.

7.   The method as claimed in claim 6, wherein the iterative process is stopped when a local minimum is found.

8.   The method as claimed in claims 6 or 7, wherein the iterative process is pre-computed and the results stored in a table for direct reference.

9.   The method as claimed in any one of claims 1 to 8, comprising
        - determining an intersection I, between the light source's frustum and the eye's frustum;
        - computing the centre point e of the vertices of I;
        - defining a centre line l_n passing through the position of the eye and e, for generating the trapezoid.

10.   The method as claimed in claim 9, further comprising defining a new focus region which lies between the near and far planes of the eye's frustum that are geometrically pushed closer to tightly bound I.

11.   The method as claimed in any one of claims 1 to 10, wherein the trapezoidal transformation comprises mapping the four corners of the trapezoid to a unit square that is the shape of a square shadow map, or to a general rectangle that is the shape of a rectangular shadow map.

12.   The method as claimed in claim 11, wherein the size of the square or general rectangle changes based on a configuration of the light source and the eye.

13.   The method as claimed in any one of the preceding claims, wherein the trapezoidal transformation transforms only the x and the y values of a vertex from the

post-perspective space of the light to the trapezoidal space, while the z value is maintained at the value in the post-perspective space of the light.

14.   The method as claimed in claim 13, comprising applying the trapezoidal transformation to obtain the x, y, and w values in the trapezoidal space, x_T, y_T, and w_T,

5   and computing the z value in the trapezoidal space, z_T, as $z_T = \frac{z_L \cdot w_T}{w_L}$, where z_L and w_L, are the z and w values in the post-perspective space of the light, respectively.

15.   The method as claimed in claim 13, comprising:

- in a first pass of shadow map generation,

> transforming coordinate values of a fragment from the trapezoidal space

10   back into the post-perspective space L of the light to obtain a first transformed fragment, utilising the plane equation of the first transformed fragment to compute a distance value of the first transformed fragment from the light source in L, z_{L1}, adding an offset value to z_{L1}, and store the resulting value as a depth value in the shadow map;

15   - in a second pass of shadow determination,

> transforming texture coordinate assigned, through projective texturing, to the fragment from the trapezoidal space back into L, obtaining a second transformed fragment from the transformed texture coordinate, utilising the plane equation of the second transformed fragment to compute a

20   distance value of the second transformed fragment from the light source in L, z_{L2}, and determine whether the fragment is in shadow based on a comparison of the stored depth value in the shadow map and z_{L2}.

16.   The method as claimed in claims 13 comprising

25   - in a first pass of shadow map generation,

> during a vertex stage, transforming coordinate values of the vertex into the trapezoidal space, and assigning to the vertex the texture coordinate equal to the vertex's coordinate values in the post-perspective space of the light, and

30   - during a fragment stage, replacing the depth of the fragment with the texture coordinate of the fragment, adding to the depth an offset, and store the resulting value as a depth value in the shadow map;

- in a second pass of shadow determination,

> - during the vertex stage, transforming coordinate values of the vertex

35   into the post-perspective space of the eye, and assigning to the vertex

two texture coordinates that are first the coordinate values of the vertex in the post-perspective space of the light and second the coordinate values of the vertex in the trapezoidal space, and

- during the fragment stage, determining shadow of the fragment based on a comparison of the stored depth value in the shadow map, as indexed based on the second texture coordinate of the fragment, with a value based on the first texture coordinate of the fragment.

17.   The method as claimed in claim 13 comprising

- in a first pass of shadow map generation,

transforming coordinate values of a fragment from the trapezoidal space back into the post-perspective space L of the light to obtain a first transformed fragment, utilising the plane equation of the first transformed fragment to compute a distance value of the first transformed fragment from the light source in L, z_{L1}, adding an offset value to z_{L1}, and store the resulting value as a depth value in the shadow map,

- in a second pass of shadow determination,

- during the vertex stage, transforming coordinate values of the vertex into the post-perspective space of the eye, and assigning to the vertex two texture coordinates that are first the coordinate values of the vertex in the post-perspective space of the light and second the coordinate values of the vertex in the trapezoidal space, and

- during the fragment stage, determining shadow of the fragment based on a comparison of the stored depth value in the shadow map, as indexed based on the second texture coordinate of the fragment, with a value based on the first texture coordinate of the fragment.

18.   The method as claimed in claim 13 comprising

- in a first pass of shadow map generation,

during a vertex stage, transforming coordinate values of the vertex into the trapezoidal space, and assigning to the vertex the texture coordinate equal to the vertex's coordinate values in the post-perspective space of the light, and

- during a fragment stage, replacing the depth of the fragment with the texture coordinate of the fragment, adding to the depth an offset, and store the resulting value as a depth value in the shadow map;

- in a second pass of shadow determination,

transforming texture coordinate assigned, through projective texturing, to the fragment from the trapezoidal space back into L, obtaining a second transformed fragment from the transformed texture coordinate, utilising the plane equation of the second transformed fragment to compute a distance value of the second transformed fragment from the light source in L, z_{L2}, and determine whether the fragment is in shadow based on a comparison of the stored depth value in the shadow map and z_{L2}.

20. The method as claimed in any one of claims 1 to 19, further comprising adding a polygon offset in the determining whether an object or part thereof is in shadow in the desired view of the scene for representation utilising the computed shadow map.

21. The method as claimed in any one of the preceding claims, wherein two or more light sources illuminate at least respective portions of the scene, and the method is applied for each light source.

22. A system for real-time shadow generation in computer graphical representation of a scene, the system comprising

- a processor unit for defining an eye's frustum based on a desired view of the scene; for defining a location of a light source illuminating at least a portion of the scene; for generating a trapezoid to approximate an area, E, within the eye's frustum in the post-perspective space of the light, L, from the light source; for applying a trapezoidal transformation to objects within the trapezoid into a trapezoidal space, for computing a shadow map; and for determining whether an object or part thereof is in shadow in the desired view of the scene utilising the computed shadow map.

23. A data storage medium having stored thereon computer code means for instructing a computer to execute a method of real-time shadow generation in computer graphical representation of a scene, the method comprising

- defining an eye's frustum based on a desired view of the scene;

- defining a location of a light source illuminating at least a portion of the scene;

- generating a trapezoid to approximate an area, E, within the eye's frustum in the post-perspective space of the light, L, from the light source;

- applying a trapezoidal transformation to objects within the trapezoid into a trapezoidal space for computing a shadow map; and

- determining whether an object or part thereof is in shadow in the desired view of the scene utilising the computed shadow map.

Annexure A

light's vertex program:
```
!!VP1.0

# c[0-3]    : N_T
# c[8-11]  : light's projection and modelview matrix
# c[12-15]: world matrix
# c[16-19]: inverse world matrix
# v[OPOS] : object position
# o[HPOS] : result vertex

# transform v[OPOS] into world space and store result in R4:
# R4 = W * v[OPOS]
DP4 R4.x, c[12], v[OPOS];
DP4 R4.y, c[13], v[OPOS];
DP4 R4.z, c[14], v[OPOS];
DP4 R4.w, c[15], v[OPOS];

# transform R4 into light's post-perspective space and store result in R1:
# R1 = P_L * C_L * (R4) = P_L * C_L * W * v[OPOS]
DP4 R1.x, c[8], R4;
DP4 R1.y, c[9], R4;
DP4 R1.z, c[10], R4;
DP4 R1.w, c[11], R4;

# store this R1 in the first texture coordinate:
# o[TEX0] = P_L * C_L * W * v[OPOS]
MOV o[TEX0], R1;

# transform R4 into trapezoidal space:
# o[HPOS] = N_T * P_L * C_L * W * v[OPOS]
DP4 o[HPOS].x, c[0], R4;
DP4 o[HPOS].y, c[1], R4;
DP4 o[HPOS].z, c[2], R4;
DP4 o[HPOS].w, c[3], R4;

MOV o[COL0], v[COL0];

END
```

light's fragment program:
```
!!FP1.0

# f[WPOS] : fragment in trapezoidal space (window position)
# f[TEX0] : fragment's position in post-perspective space of the light

RCP R0.x, f[TEX0].w;        # R0.x = 1 / w_L
MUL R1, f[TEX0], R0.x;      # R1 = (x_L/w_L, y_L/w_L, z_L/w_L, 1)
MAD R2.z, R1.z, 0.5, 0.5;   # R2.z = R1.z * 0.5 + 0.5; depth is now in
                            # the range [0;1]
MUL o[DEPR], R2.z, R2.z;    # o[DEPR] = z_L/w_L * 0.5 + 0.5; replace
                            # "z_T" with "z_L"
MOV o[COLR], f[COL0];

END
```

Annexure B

```
void display() {

// 1st pass: TSM generation

// as described with reference to Figure 6 in the detailed description,
// as a first step we have to calculate a trapezoid based on the
// eye frustum E.
// The four vertices are stored in t_0, t_1, t_2, t_3
calculateTrapezoid(&t_0, &t_1, &t_2, &t_3, P_L, C_L, E);

// ...after that N_T is calculated as described above
calculateTrapezoidalTransformation(&N_T, t_0, t_1, t_2, t_3);

glViewport(0, 0, shadowmapWidth, shadowmapHeight);

// Bind the above vertex program...
glBindProgramNV(GL_VERTEX_PROGRAM_NV, vpLight);

// ...and bind the above fragment program
glBindProgramNV(GL_FRAGMENT_PROGRAM_NV, fpLight);

glMatrixMode(GL_PROJECTION); // store N_T in GL_PROJECTION
glLoadMatrixf(N_T);
glTrackMatrixNV(GL_VERTEX_PROGRAM_NV, 0, GL_PROJECTION, GL_IDENTITY_NV);

glMatrixMode(GL_MATRIX1_NV); // store in P_L * C_L and track in GL_MATRIX1_NV
glLoadMatrixf(P_L); // light's projection matrix, e.g. achieved with
                    // gluPerspective()
glMultMatrixf(C_L); // light's camera matrix, e.g. achieved with
                    // gluLookAt()
glTrackMatrixNV(GL_VERTEX_PROGRAM_NV, 8, GL_MATRIX1_NV,GL_IDENTITY_NV);

glMatrixMode(GL_MODELVIEW);  // store the modelview matrix in
                             // GL_MODELVIEW
glLoadIdentity();
glTrackMatrixNV(GL_VERTEX_PROGRAM_NV, 12, GL_MODELVIEW,GL_IDENTITY_NV);

renderScene();

// Like in the standard shadow map approach we copy the depth buffer
// into a texture:
CopyDepthBufferToTexture();

        ...


// 2nd pass: render scene from eye's position and project TSM texture
// over the scene
        ...

SwapBuffers();

}
```